Mathematics, engineering and its technology

Abstract. Mathematicians have determined that there is a set of problems that cannot be solved because they could solve a problem that is still open. However, the expectations of logical formalisms sometimes do not go hand in hand with how engineering works. So if they are wrong, they condemn the rest of the sciences to be unable to develop a technology that is possible. We will therefore refute Cook's theorem so that it does not continue to be a burden for technology.

Keywords: NP, P-class, TM, Complexity, philosophy, formalism

Introduction

It is very difficult to determine who was the first person to develop a philosophy of technology. We may well attribute it to Aristotle [1], when he discovered the difference between material and non-substantial implication in his explanations of metaphysics. Likewise, Plato already distinguished ideas that were true in themselves from those that were the product of human hands.

Distinguishing when a statement is governed by a certain logic is fundamental to understanding when we can be sure of a statement that cannot actually be true. This is something that Kant himself criticised in the “Critique of pure reason” [2]: it was due to mixing ideas of a certain nature with a method inappropriate to that nature. That is, what you experience you must synthesise and what you conceptualise you must analyse. Likewise, in his “Critique of Practical Reason” [3], Kant makes a profound reflection on mathematics, in order to observe a double behaviour in it. That is to say, for Kant there is mathematics that is discovered and mathematics that is formed.

Following the chronological order, it seems obligatory to mention Ada Lovelace, who was perhaps the first mathematician to see in numbers the symbolic information necessary to represent a much more practical technology [4]. When the Pythagoreans would have killed a wayward who denied that the square root of two was a rational number, Ada would observe in roots objects that would transcend the idea of number: they could be enumerable expressions, which goes further.

With the arrival of Cantor, numbers began to have a form established by the mathematicians themselves, the process of discovering how naturals behave ceased to be of interest, and there was enormous disenchantment because the nature of numbers was

no longer unique. There were two infinities, potential infinity (which was just a number large enough to make any other quantity negligible) and formal infinity (which, being simply infinite, could be defined in an infinite number of ways).

With the new formalisms came a range of problems, so Bertrand Russell and Whitehead created the Principia Mathematica [5] in order to find out how knowledge should be inferred. However Gödel showed that formal logic combined with Peano arithmetic and a process of creating theorems leads to an incoherence if the theory were complete.

Thus we come to Turing, who presents us with a notation and a set of problems that although they can be formally defined, they are technologically impossible to set up [6]. These problems are called undecidable, like the halting problem. And it is the first rigorous demonstration that technology (what is decidable) works under a different language than logical formalisms (what is enumerable). You cannot develop everything that is logically possible.

That is, we can create a logical theory and mathematical formalisms full of operations, but that is not going to tell us much about the limits of technology because it has a different nature.

Popper warned us that there were two different worlds in the world of ideas [7]: the cultural and the discoverable. It is therefore not surprising that among the cultural there is a large number of ideas as perfect as a logical formalism and that, not being connected with the world of the discoverable, this would lead us to the existence of another world of ideas outside cultures.

Being two worlds will mean that ideas will have to develop in two different ways.

Crossing the two worlds

A Turing machine is a mathematical model of computation that defines an abstract machine that manipulates symbols on a strip of tape according to a table of rules. It is considered that in that machine it is possible to denote any enumerable problem, and when each of the intermediate states are exactly determined Alan Turing designated that as an automatic Turing Machine [6] (a-TM, deterministic TM), if, on the contrary, it is always possible that the next state can be a finite number of possible states then that machine is denoted a choice Turing Machine (c-TM, non-deterministic TM, NDTM).

The Boolean satisfiability problem (sometimes called propositional satisfiability problem

and abbreviated SATISFIABILITY or SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. That problem is usually reduced to formulas which are a Boolean product of sum of literals. Where a literal is a Boolean variable negated or not, and a sum of literals is called a clause.

If we write on a tape n symbols as the expression of a product of clauses and ask what values the variables must take to satisfy the formula, if we had the result we would only need polynomial time (the time is expressed less than k^n for any k) to determine if the formula is satisfied. When a configuration of a TM is solved in polynomial time the problem is considered in class P, and when it is solved in NDTM then in NP class. Therefore, the SAT problem is considered trivially within the NP class. Having a mechanism that allows us to know what values the variables should take in polynomial time would convert SAT to the class P, for more details [8] or [9].

With respect to SAT, Cook warned that by solving SAT all the problems of the NP class would be solvable in polynomial time [10]. To prove it, Cook set up a Turing machine so that it generated a SAT input deterministically in polynomial time, you can find that demonstration in [11]. The real problem is not only that it may be solved in polynomial time, but also if for any practicable size n: $n^k < 2^{309}$, which corresponds to a transcomputational limit that is too many combinations even for the fastest machines [12].

When Stephen Cook began to mix the two worlds, he did so by means of a theorem, and in the same way, Fagin reaffirmed that hypothesis by developing the theory of NP and P classes from a formal point of view by finding equivalence with second-order logic [13]. However, these authors did not realise the simplicity of these affirmations: can logic explain the functioning of a machine that completes all mathematics? Does this not refute Gödel?

As if this were not enough, Matiyasevich presented a result [14]: it was not possible to know whether two finite expressions of sums and products of variables and constants (diophantine equations) were equal, because a Turing machine could be simulated by means of diophantine equations. Thus, not only did he reaffirm Turing's work, but he also made use of a result by Julia Robinson by means of which, with a limited number of sums and products, he could simulate an exponential.

Now, if you simulate an exponential with sums and products, this means that exponential expressions have a polynomial representation, without palliatives formally NP = P. Indeed,

as Matiyasevich himself explains, Robinson did not actually find the constructive solution, but a sufficient condition. That is, a formalism. This formalism stated that there must be a sufficiently large integer to satisfy a condition. Such a condition can always exist, but there will always be entries where it does not. That is the difference between the constructivist world and the formal world. That is the nuance between the two worlds.
Formally it is possible to speak of a technology that approximates to what logic establishes, but constructively it is impossible.
There are authors who do not understand these approaches, that is why they do not understand that it is considered possible that the definition of Turing Machine is independent of whether the class P is equivalent to the class NP. Without going any further, we know that a looser definition of the Turing machine offers the possibility of solving undecidable problems [15], so if we change the notation it is not surprising that there are complexity classes that were different before and now are the same.
However, it is important to know that there are techniques that allow us to solve SAT, and if the bridge established by Cook were used, then the Turing machine could well be used to propose innovations in various fields such as medicine, pharmaceuticals, etc... This is important from a practical point of view, because just because worst-case complexity leads to a combinatorial explosion does not mean that it cannot be put to practical use. In fact, this is seen as very common [16]. Likewise, if what Fagin tells us were true, then we could use his technology as a bridge to program through PROLOG all the innovations that represent the most important management problems, as presented in this book [11]. But no matter how many decades have passed, we are not aware of the existence of such bridges, despite the enormous convenience of doing so, including saving lives through COVID by solving QAP (Quadratic Assignment Problem explained in [11]).
It should be obvious that if there are no libraries using Cook's or Fagin's results, despite the enormous economic convenience, it is because such a connection between the two worlds does not really exist.

Operational Vs Formal

An operation (without grammatical constraints) causes a succession of experiences in the machine, while a syllogism analyses the state in which the system is.

We are going to have a Turing machine that transforms states over time thanks to a configuration σ. The logical formalism tells us that $Q_a \vdash Q_b$ if and only if there exists an integer K where $Q_{a,k} \rightarrow Q_{b,k+1}$, where → is the logical implicator. Moreover we know that the fact that $Q_a \vdash Q_b$ in the Turing machine implies a that the transition (a, b) is described in

σ. Being a and b two indices to complete states of a Turing machine, not only its register. So we start from a Turing machine that intends to solve a problem that will end in two possible final states: $Q_{Y, T}$ or $Q_{N, T}$ at some T for each input.

1. We define $Q_{z, k}$ a state through which the input to which we submit the machine in time k does not pass and which leads to the rejection $Q_{N, k+1}$ directly.
2. (z, N) is transition described in σ according to 1.
3. For all k: $Q_{z, k} \rightarrow Q_{N, k+1}$ [It follows from 2 and 1, leads to rejection].
    4. $(Q_{z, k} \rightarrow Q_{Y, k+1})$ OR $(Q_{Y, k+1} \rightarrow Q_{z, k})$ [theorem According to [17]].
        1. Sup $Q_{z, k} \rightarrow Q_{Y, k+1}$
        2. Formally we deduce that $Q_{z, k}$ implies two different states in a deterministic machine. Both $Q_{Y, k+1}$ and $Q_{N, k+1}$.
5. $Q_{Y, k+1} \rightarrow Q_{z, k}$ [It follows from the previous branch, by contradiction].
6. $Q_{Y, k+1} \rightarrow Q_{N, k+1}$ [Combining steps 3 and 5].
7. So if a state is recognised it is also not recognised at the same time.

Cook's theorem [11] is therefore refuted and the formalisms are too lax to represent machines. Strict logic is a blow to our instincts when we design machines and try to pigeonhole them into logical models.

Conclusions

It is extremely dangerous to turn a logician into a technology consultant, and a good example of this is when they limit your ability to design proteins [18]. In order to deny the existence of a technology, one must use the principles of decidability, but once this is done, there will still be more imprecise machines that will give approximate results. That is, as soon as we deny that transcendental numbers exist, we can always admit Cantor's formalisms as a precursor of differential equations and the birth of a technology that, although not very precise, has been a great advance for science.

References

1. Aristotle Organon And Other Works: https://archive.org/details/AristotleOrganon/mode/2up
2. *Kant, Immanuel (1999). Critique of Pure Reason. The Cambridge Edition of the Works of Immanuel Kant. Translated by Guyer, Paul; Wood, Allen W. Cambridge: Cambridge U.P.*
3. https://librivox.org/critique-of-practical-reason-by-immanuel-kant/
4. *"Sketch of The Analytical Engine, with notes upon the Memoir by the Translator". Switzerland: fourmilab.ch. October 1842. Retrieved 28 March 2014.*

http://www.fourmilab.ch/babbage/sketch.html

5. https://plato.stanford.edu/entries/principia-mathematica/

6. Turing, A. M. . On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society,* **S2-42**(1), 230-265. doi:10.1112/plms/s2-42.1.230 (1937)

7. Popper, Karl Three Worlds by Karl Popper. The Tanner Lecture on Human Values. Talk delivered at The University of Michigan (7 April 1978).

8. Arora, S. and Barak, B. Computational complexity. *New York: Cambridge University Press.* (2016)

9. Karp, R. M. Reducibility among Combinatorial Problems. *Complexity of Computer Computations,* 85-103. doi:10.1007/978-1-4684-2001-2_9 (1972)

10. Cook, S. The importance of the P versus NP question. *Journal of the ACM,* **50**(1), 27-29. doi:10.1145/602382.602398 (2003)

11. Garey, M. R., & Johnson, D. S. *Computers and intractability: A guide to the Theory of NP-Completeness*. New York, NY: W.H. Freeman. (2009)

12. Miles, William. “Bremermann’s Limit”. Retrieved 1 May 2011.

13. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. Complexity of Computation, ed. *R. Karp, SIAM-AMS Proceedings* **7**, pp. 27–41. 1974.

14. Matijasevich, Y. . My collaboration with Julia Robinson. *The Mathematical Intelligencer,* **14**(4), 38-45. doi:10.1007/bf03024472 (1992)

15. Abdulla, P. A., & Jonsson, B. Verifying Programs with Unreliable Channels. *Information and Computation,* **127**(2), 91-101. doi:10.1006/inco.1996.0053 (1996).

16. Spielman, D. A., & Teng, S. Smoothed analysis. *Communications of the ACM,* **52**(10), 76-84. doi:10.1145/1562764.1562785 (2009)

17. https://plato.stanford.edu/archives/sum2009/entries/logic-relevance/

18. Pierce NA, Winfree E. Protein design is NP-hard. Protein Eng. 2002 Oct;15(10):779-82. doi: 10.1093/protein/15.10.779. PMID: 12468711.